Assignment1 - Architecture

Alex Enze Liu

February 2018

1 README

1.1 Submission

The source code is in the "src" directory. There are two files.

- One file is called "CacheEmulator.py", which contains the python implementation of a cache.
- Another file is called "CacheSimulation.py", which is used for generating test results.

The results by executing "CacheSimulation.py" are in "graphs" directory. There are 12 graphs, 4 for each algorithm

1.2 Running The Code

Langauge:

- Python3.6

Dependencies:

- Matplotlib
- Numpy

Running Configurations:

- The default running cache size is set to 1024 if we're not varying the cache size. (The default cache size in program when reading parameters is still 65536).
- Other default running settings are the same as specified in "project1.pdf".
- When using different cache associativities (or block sizes, etc.), other settings are set to default (i.e. when test different associativities, other settings are set to default).

- Vector Size: 20000, Matrix Size: 100 * 100, Block Size: 10.

Running The Code:

- To get test results for all three algorithms, type:
  "python3 CacheSimulation.py"
- To test cache, type:
  "python3 CacheEmulator.py"

1.3 Code Description

A cache is implemented in "CacheEmulator.py".

It has two global variables: "logging" and "conf". "logging" is used to log statistics like read miss or read hit. "conf" stores the program running configuration.

It then implements several classes: "DataBlock", "CPU", "Cache", "RAM". The implementation extends the specification in "Project1.pdf".

Next, it defines three functions for three algorithms, "dot", "mxm" and "mxm_block". The logic behind these functions are as follows:

- Define address for arrays/vectors.
- Initializing arrays/vectors in RAM without using CPU, so the cache is clean.
- Run a bunch of pseudo-assembly operations to perform three algorithms.
- Check the final result stored in RAM is correct as expected.

Finally it defines a main function that parses the args, and call the corresponding functions. In the end print out the statistics.

2 Analysis

2.1 DOT

The results for DOT algorithm are displayed below in figure 1, 2, 3 and 4.

It can be seen from the figures that under the assumption that the vector size is much larger than cache size, only changing block size will influence hit rate and miss rate. It makes sense because dot operation does not reuse any data, so miss rate and hit rate is only affected by block size, as it decides the number of doubles read into cache each time.

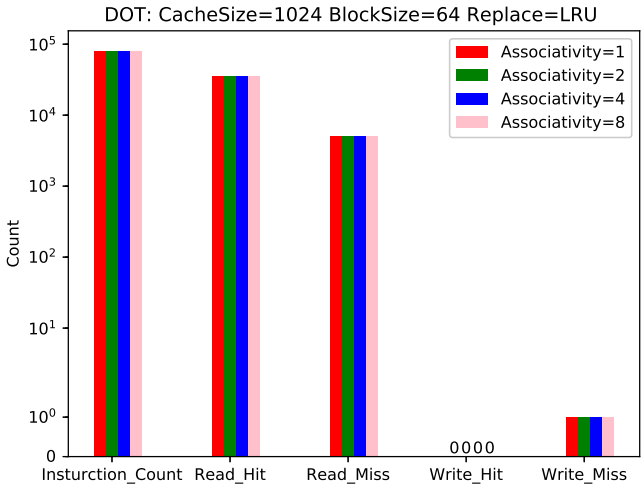

Figure 1: Different Associativity

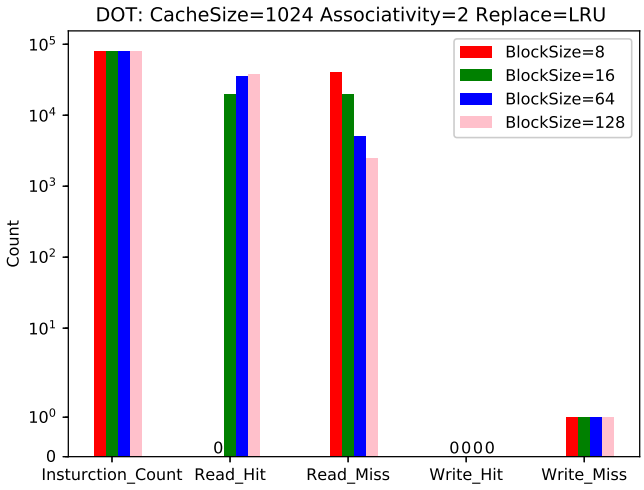

Figure 2: Different Block Size

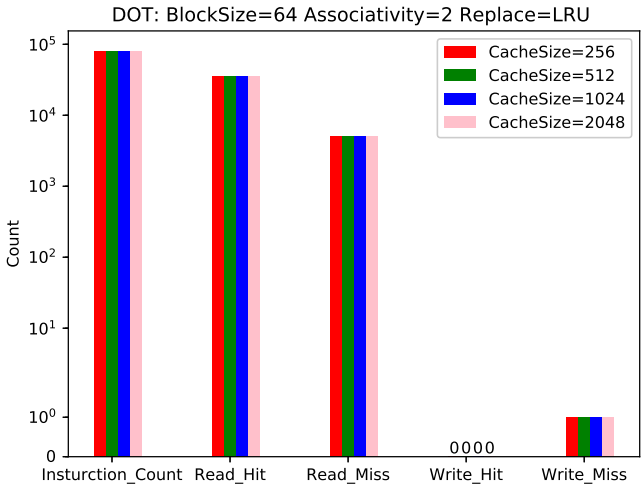

Figure 3: Different Cache Size

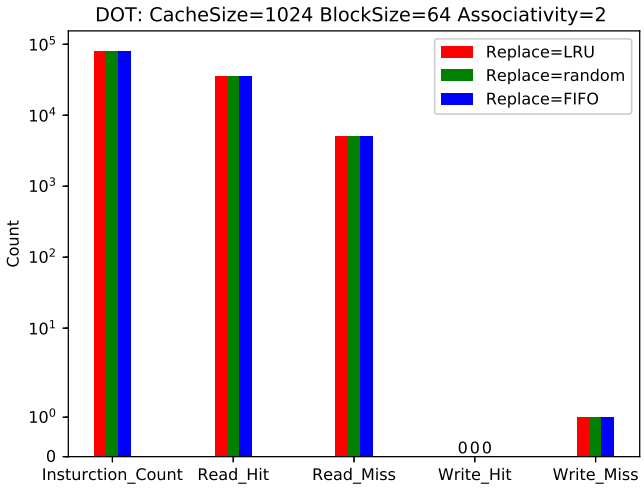

Figure 4: Different Replacement Algorithm

2.2 MXM AND MXM_BLOCK

The results for MXM and MXM_BLOCK algorithms are displayed below in figure 5, 6, 7, 8, 9, 10, 11 and 12.

In general, we can see that MXM_BLOCK outperforms MXM algorithm in hit rate under different settings. We also see that the write hit rate is 0 in most cases in MXM. However, in MXM_BLOCK, we see that the hit rate is around 50% in most cases. It's exactly what we expected because MXM_BLOCK make full use a block of data each time, so hit rate are likely to increase a lot. We are also able to observe what we see in DOT algorithm - block size affects hit rate. It's also worth mentioning that as can be seen from figure 8 and 12, FIFO and LRU are slight better than random. The reason behind it might be that even though FIFO and LRU are making use of temporal locality, the block size is still larger than cache. Thus, the performance improvement of FIFO and LRU is limited.

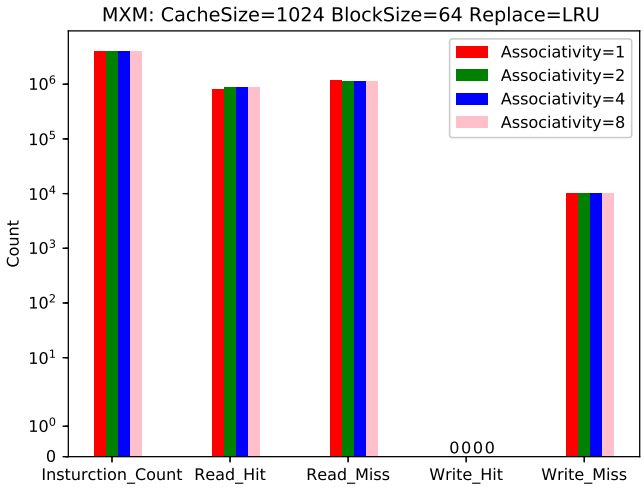

Figure 5: Different Associativity

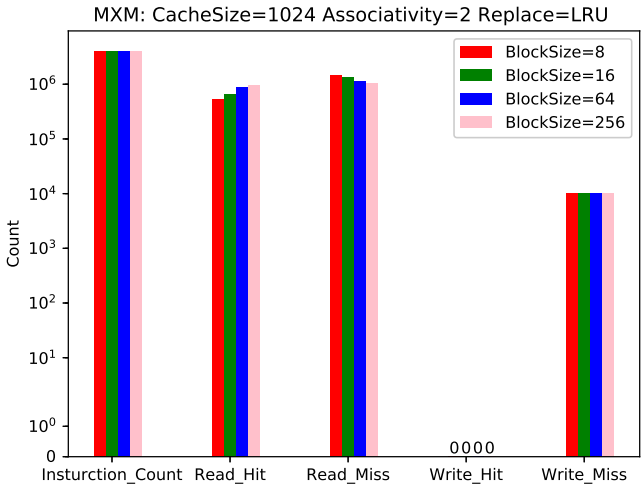

Figure 6: Different Block Size

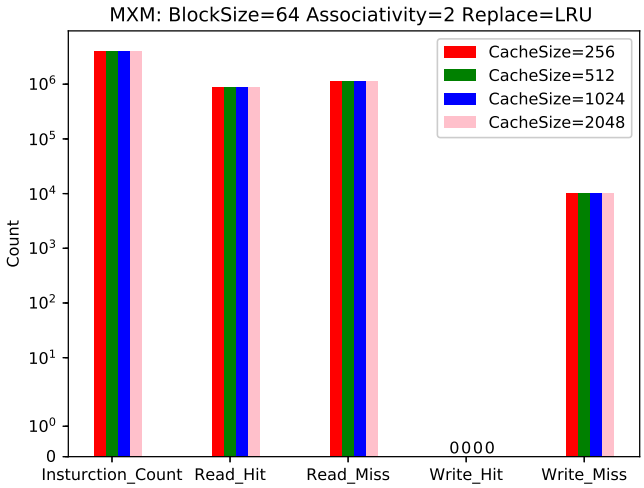

Figure 7: Different Cache Size

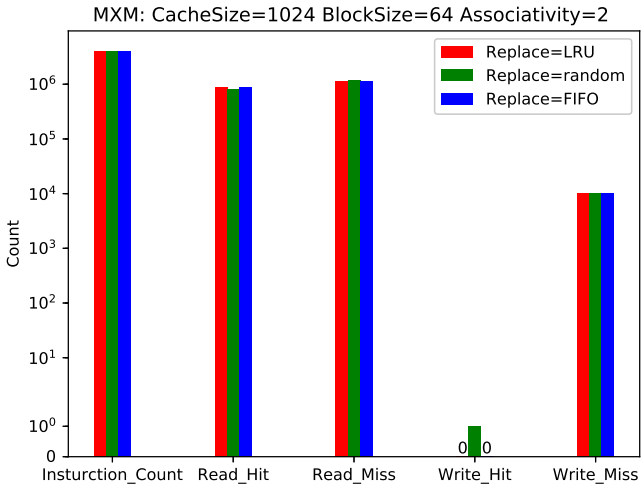

Figure 8: Different Replacement Algorithm

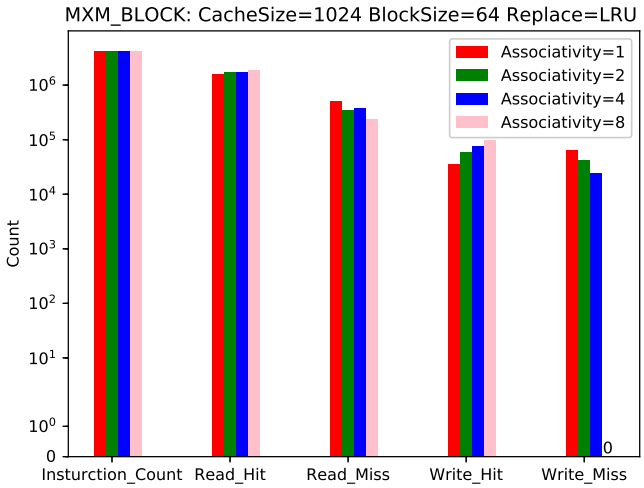

Figure 9: Different Associativity

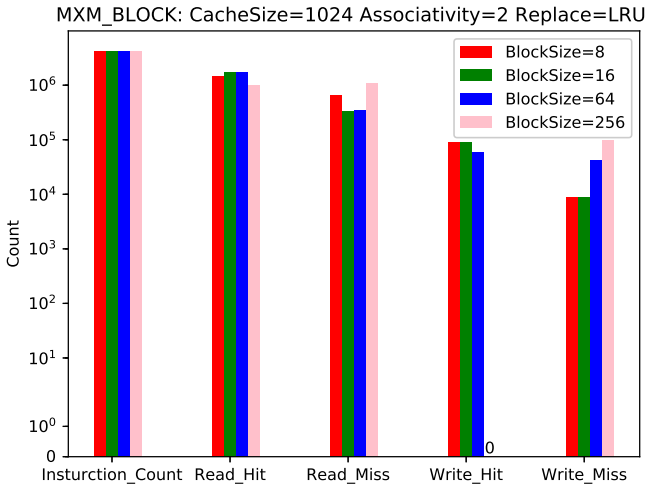

Figure 10: Different Block Size

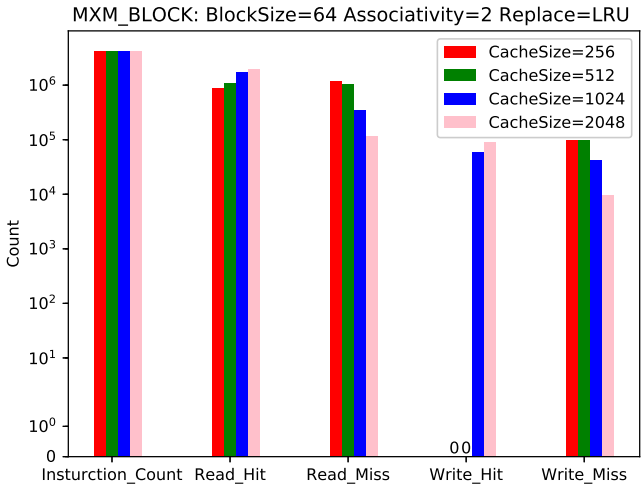

Figure 11: Different Cache Size

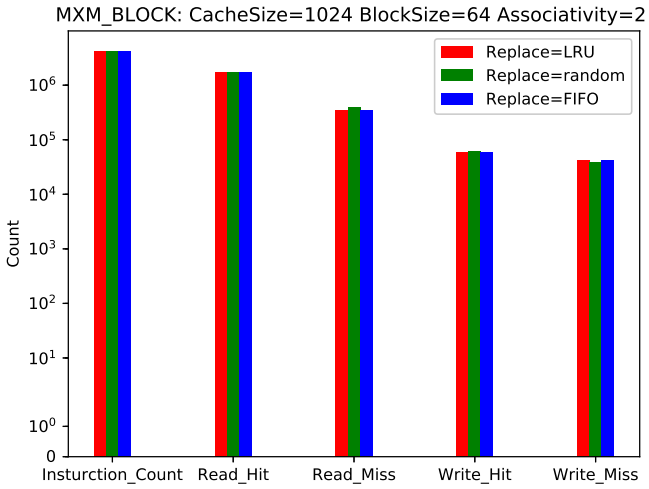

Figure 12: Different Replacement Algorithm